Introduction to Propositional Logic

Jinbo Huang

Australian National University

Overview

- Syntax and semantics
- Reasoning methods

Propositional Logic: Ingredients

$Burglary \vee Earthquake \rightarrow Alarm$

Variables

- $A, B, C, \ldots$
- $Burglary, Earthquake, Alarm, \ldots$

Connectives

- $\neg$ (not, negation)
- $\wedge$ (and, conjunction)
- $\vee$ (or, disjunction)
- $\rightarrow$ (implication)
- $\leftrightarrow$ (biconditional)

Propositional Logic: Sentences

Every variable is a sentence

If α and β are sentences, so are

- $\neg(\alpha)$
- $(\alpha) \wedge (\beta)$
- $(\alpha) \vee (\beta)$
- $(\alpha) \rightarrow (\beta)$
- $(\alpha) \leftrightarrow (\beta)$

Parentheses omitted where no confusion arises

Propositional Sentences

$$Burglary \lor Earthquake \rightarrow Alarm$$

$$(A \lor B) \land (A \lor \neg B) \rightarrow A$$

$$\neg(A \land B) \leftrightarrow (\neg A \lor \neg B)$$

Semantics: Worlds

World, interpretation, evaluation, truth assignment, variable instantiation

- Map from variables to {true, false} / {1, 0}
- $\omega_1(A) = 1, \omega_1(B) = 0, \omega_1(C) = 1, \ldots$
- $\omega_2(A) = 0, \omega_2(B) = 1, \omega_2(C) = 1, \ldots$
- 2^n possible worlds for n variables

Semantics: Truth of Sentences, Models

Given world ω, $\omega \models \alpha$: α evaluates to 1 under ω

- $\omega \models X$ iff $\omega(X) = 1$
- $\omega \models \neg\alpha$ iff $\omega \not\models \alpha$
- $\omega \models \alpha \wedge \beta$ iff $\omega \models \alpha$ and $\omega \models \beta$
- $\omega \models \alpha \vee \beta$ iff $\omega \models \alpha$ or $\omega \models \beta$
- $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$
- $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

$\omega \models \alpha$: ω is a *model* of α, ω *satisfies* α

Truth of Sentences

$Burglary \vee Earthquake \rightarrow Alarm$

- $\omega_1 = \{Burglary, \overline{Earthquake}, Alarm\}$
- $\omega_2 = \{Burglary, Earthquake, \overline{Alarm}\}$

Sentences as Boolean Functions

Each sentence is a function

- Plug in values (1/0) for variables
- Get value (1/0) for whole sentence

Function depends only on the mapping, not composition of sentence

Different sentences can define same function

- $A \vee (B \wedge C)$
- $(A \vee B) \wedge (A \vee C)$

Sentences as Boolean Functions

Set of models determines function

For n variables

- # possible sentences infinite
- # possible worlds 2^n
- # possible sets of models 2^{2^n}
- # possible functions 2^{2^n}

Completeness of Connectives

Is every Boolean function represented by some sentence?

Depends on the connectives allowed

Not if only $\{\wedge, \vee\}$ allowed, e.g.

- Cannot express $\neg A$
- Proof by noting monotonicity

Completeness of Connectives: $\{\wedge, \vee, \neg\}$

Function $\equiv$ set of models

$\{\overline{A}BC, A\overline{B}C, ABC\}$

- $\neg A \wedge B \wedge C$
- $A \wedge \neg B \wedge C$
- $A \wedge B \wedge C$

$(\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$

Set of models $\Rightarrow$ sentence using only $\{\wedge, \vee, \neg\}$

Completeness of Connectives: $\{\wedge, \neg\}$

Can express $\vee$ using $\{\wedge, \neg\}$

$A \vee B \equiv \neg(\neg A \wedge \neg B)$

Similarly, $\{\vee, \neg\}$ is complete

$A \wedge B \equiv \neg(\neg A \vee \neg B)$

Completeness of Connectives: $\{\uparrow\}$

NAND: $A \uparrow B \overset{def}{=} \neg(A \land B)$

$\land, \lor, \neg$ can be expressed with $\uparrow$ alone

- $A \land B \equiv (A \uparrow B) \uparrow \top$
- $A \lor B \equiv (A \uparrow \top) \uparrow (B \uparrow \top)$
- $\neg A \equiv A \uparrow \top$

$\top$: true

$\bot$: false

Similarly, $\{\downarrow\}$ (NOR alone) is complete

Useful Identities

DeMorgan's

- $\neg(A \wedge B) \equiv \neg A \vee \neg B$
- $\neg(A \vee B) \equiv \neg A \wedge \neg B$

Distribution

- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Reasoning Tasks

Satisfiability (SAT)

- Given α, does α have a model?
- $\exists \omega, \omega \models \alpha$

Validity

- Given α, is α 1 in all worlds?
- $\forall \omega, \omega \models \alpha$
- Reducible to SAT: α is valid iff $\neg\alpha$ is not SAT

Reasoning Tasks

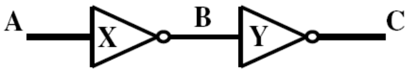
Entailment

- Given α and β, is β 1 whenever α is 1?
- $\alpha \models \beta$: models(α) $\subseteq$ models(β)
- Reducible to validity: $\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid

Equivalence

- $\alpha \equiv \beta$: models(α) $=$ models(β)
- Reducible to entailment: $\alpha \models \beta$ and $\beta \models \alpha$

How is Logic Useful?



Knowledge base Δ

- $okX \rightarrow (A \leftrightarrow \neg B)$
- $okY \rightarrow (B \leftrightarrow \neg C)$

Is $A\overline{C}$ normal?

- Is it possible for okX and okY to be both 1?
- Is $(A \wedge \neg C \wedge okX \wedge okY \wedge \Delta)$ SAT?

How is Logic Useful?

Diagnosis

- Is observation normal?
- What components might be broken?

Verification

- Does circuit/program correctly implement specification?
- Can some “bad” behavior happen?

Planning

- Is there sequence of actions to reach goal?

Reasoning Methods: Is α SAT?

Semantic reasoning

- Search for ω such that $\omega \models \alpha$

Syntactic reasoning (inference, proof)

- Create new sentences from α by applying syntactic rules
- Stop when proof of UNSAT is produced
- Rules should be sound and complete
  - UNSAT proof will be produced iff α is not SAT

Negation Normal Form (NNF)

$\{\wedge, \vee, \neg\}$ only

$\neg$ applied to variables only

NNF

- $(A \wedge \neg B) \vee C$
- $(A \vee B) \wedge (B \vee \neg C)$

Not NNF

- $A \rightarrow (B \vee C)$
- $\neg(\neg(A \wedge B) \wedge (C \vee D))$

Conversion to NNF

Eliminate connectives other than $\wedge, \vee, \neg$

- Always possible by completeness of $\{\wedge, \vee, \neg\}$

Push $\neg$ all the way to variables

- Always possible by DeMorgan's laws

$\neg(\neg(A \wedge B) \wedge (C \vee D))$ becomes

$(A \wedge B) \vee (\neg C \wedge \neg D)$

Conjunctive Normal Form (CNF)

Literal: variable or its negation

- $A, \neg A, B, \neg B, C, \neg C, \ldots$

Clause: disjunction ($\vee$) of literals

- $A \vee \neg B \vee C$
- $A, \neg B, C$ (*unit* clauses)

CNF: conjunction ($\wedge$) of clauses

- Often regard as set of clauses

Conversion to CNF

Start with NNF

Push $\vee$ all the way to literals

- Always possible by distribution laws

$(A \wedge B) \vee (\neg C \wedge \neg D)$ becomes

$(A \vee \neg C) \wedge (B \vee \neg C) \wedge (A \vee \neg D) \wedge (B \vee \neg D)$

Potential issue?

Conversion to CNF: Complexity

Can get too many clauses

- Can be exponential in size of original formula

$CNF_1 \vee CNF_2$

- Result has $|CNF_1| \cdot |CNF_2|$ clauses

$CNF_1 \vee CNF_2 \vee \ldots \vee CNF_n$

- If CNF_i each has 2 clauses, result has 2^n clauses

Exponential worst case inevitable

Conversion to CNF: Polynomial Method

Do not insist on equivalence

Convert Δ to Δ'

- Δ' is in CNF
- Δ' has extra variables
- $|\Delta'|$ is polynomial in $|\Delta|$

How can Δ' be useful if $\Delta' \not\equiv \Delta$?

- All we need is $\text{SAT}(\Delta) \equiv \text{SAT}(\Delta')$

Conversion to CNF: Polynomial Method

$\Delta = \Delta_1 \vee \Delta_2$, assume Δ_1, Δ_2 already in CNF

- $\Delta_1 = g_1 \wedge g_2 \wedge \ldots \wedge g_p$
- $\Delta_2 = h_1 \wedge h_2 \wedge \ldots \wedge h_q$

Original conversion of $\Delta_1 \vee \Delta_2$ produces pq clauses

Introduce new variable X, let Δ' have following clauses

- $X \vee g_1, X \vee g_2, \ldots, X \vee g_p$
- $\neg X \vee h_1, \neg X \vee h_2, \ldots, \neg X \vee h_q$

$|\Delta'| = p + q$

Conversion to CNF: Preservation of SAT

Δ' with new variable X

- $X \vee g_1, X \vee g_2, \ldots, X \vee g_p$
- $\neg X \vee h_1, \neg X \vee h_2, \ldots, \neg X \vee h_q$

$\text{SAT}(\Delta = \Delta_1 \vee \Delta_2) \Rightarrow \text{SAT}(\Delta')$

- $\exists \omega, \omega \models \Delta$
- $\omega \models \Delta_1$ or $\omega \models \Delta_2$, assume former
- $\forall i, \omega \models g_i$
- Let $\omega'(X) = 0$ and $\omega' = \omega$ otherwise
- $\omega' \models \Delta'$

Conversion to CNF: Preservation of SAT

Δ' with new variable X

- $X \vee g_1, X \vee g_2, \ldots, X \vee g_p$
- $\neg X \vee h_1, \neg X \vee h_2, \ldots, \neg X \vee h_q$

$\mathrm{SAT}(\Delta') \Rightarrow \mathrm{SAT}(\Delta = \Delta_1 \vee \Delta_2)$

- $\exists \omega', \omega' \models \Delta'$
- $\omega'(X) = 0$ or $\omega'(X) = 1$, assume former
- $\forall i, \omega' \models g_i$
- $\omega' \models \Delta_1$
- $\omega' \models \Delta$

Conversion to CNF

Any sentence can be put in CNF (using extra variables) in polynomial time preserving SAT

Will assume input to reasoning task is in CNF

Reasoning Methods: Is α SAT?

Semantic reasoning

- Search for ω such that $\omega \models \alpha$

Syntactic reasoning (inference, proof)

- Create new sentences from α by applying syntactic rules
- Stop when proof of UNSAT is produced
- Rules should be sound and complete
  - UNSAT proof will be produced iff α is not SAT

Syntactic Reasoning: Resolution

$$\frac{A \lor \neg B \lor C, \;\; \neg C \lor D \lor \neg E}{A \lor \neg B \lor D \lor \neg E}$$

$$\frac{g \lor X, \;\; \neg X \lor h}{g \lor h}$$

New clause called *resolvent*

If $g = h = \emptyset$, empty clause $\emptyset$ is generated

Resolution

Repeat until saturation or generation of $\emptyset$

- Will always terminate as $\#$ clauses is finite

Sound and complete

- CNF Δ is SAT iff $\emptyset$ is not generated

Resolution: Soundness

Generation of $\emptyset \Rightarrow$ UNSAT(Δ)

$$\frac{g \vee X,\ \neg X \vee h}{g \vee h}$$

- $(\omega \models g \vee X)$ and $(\omega \models \neg X \vee h) \Rightarrow (\omega \models g \vee h)$
- $(\omega \models \Delta) \Rightarrow (\omega \models$ all added clauses$)$
- $(\omega \models X)$ and $(\omega \models \neg X)$ impossible
- SAT(Δ) $\Rightarrow$ no $\emptyset$

Resolution: Completeness

$\text{UNSAT}(\Delta) \Rightarrow$ generation of $\emptyset$

No $\emptyset \Rightarrow \text{SAT}(\Delta)$

Consider all clauses at termination, repeat for $i = 1, \ldots, n$

- Set $\omega(X_i)$ to either $1/0$ without any clause being falsified by assignments so far
- Will always succeed

$\omega \models$ all clauses; hence $\text{SAT}(\Delta)$

Resolution: Completeness

Can always set $\omega(X_i)$ to 1/0 without any clause falsified by assignments so far, because otherwise

- $\exists\ X_i \vee g$, $\neg X_i \vee h$, g and h are falsified
- But $g \vee h$ exists (due to saturation), so failure must have occurred earlier
- Base case is the first iteration ($i = 1$), which cannot fail because there are no X and $\neg X$

Resolution

Will always prove SAT/UNSAT

Purely syntactic, not directly concerned with existence of models

How well does it work in practice?

Resolution: Practical Considerations

Can generate exponential # resolvents

Many may not contribute to proof (i.e., $\emptyset$)

How do we decide which resolvents to generate?

Let's come back to this issue

Reasoning Methods: Is α SAT?

Semantic reasoning

- Search for ω such that $\omega \models \alpha$

Syntactic reasoning (inference, proof)

- Create new sentences from α by applying syntactic rules
- Stop when proof of UNSAT is produced
- Rules should be sound and complete
  - UNSAT proof will be produced iff α is not SAT

Semantic Reasoning: Searching for Model

$\{\overline{A} \vee B, \overline{B} \vee C\}$ (alternative notation $\overline{A}$ for compactness)

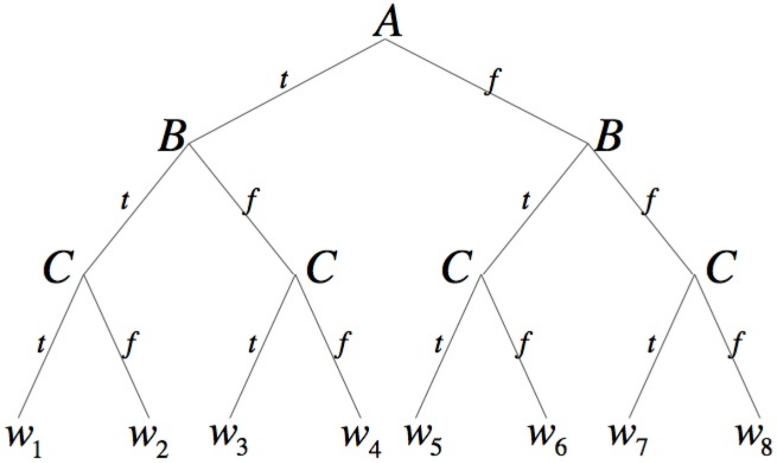
Is formula SAT?

8 assignments to ABC: enumerate & check, until model found, e.g., $A = B = C = 1$

Exponential in # variables in worst case (fine, but can try to do better in average case)

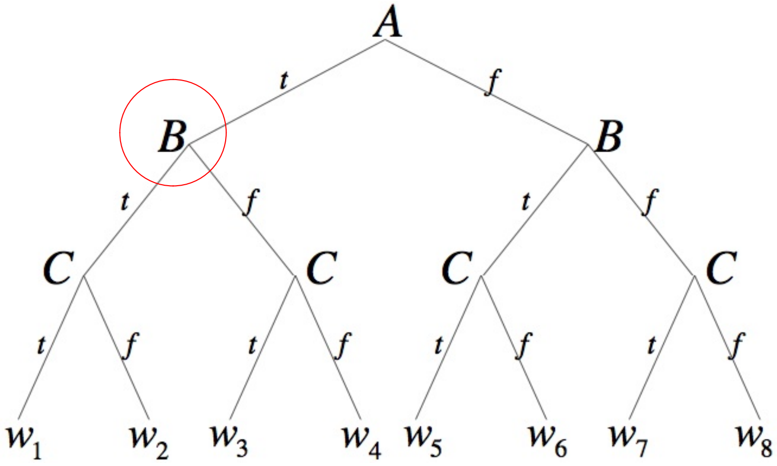
Search

$\{\overline{A} \vee B, \overline{B} \vee C\}$



Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}|_B \quad \{C\}$



Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}|_B \quad \{C\}$

Simplifies formula

- false literal disappears from clause
- true literal makes clause disappear

Leaves of search tree

- empty clause generated: formula falsified
- all clauses gone: formula satisfied

Not necessarily 2^n leaves, even in case of UNSAT

Early Backtracking

Backtrack as soon as empty clause generated

Can we do even better?

What about unit clauses?
$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \ \ \{B, \overline{B} \vee C\}$

B must be true, no need for two branches

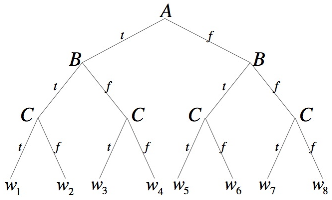
Setting $B = 1$ may lead to more unit clauses, repeat till no more (or till empty clause)

Known as unit propagation

Algorithm: DPLL

Same kind of search tree, each node augmented with unit propagation

- multiple assignments in one level
    - decision & implications
- may not need n levels to reach leaf



What (completely) determines search tree?

DPLL: Variable Ordering

Can have huge impact on efficiency

Example: unit propagation lookahead, as in SATZ

- short clauses are good, more likely to result in unit propagation
- tentatively try each variable, count new binary clauses generated
- select variable with highest score: $w(X) \cdot w(\overline{X}) \cdot 1024 + w(X) + w(\overline{X})$

Generally different orders down different branches: dynamic ordering

Enhancement

Given variable ordering, search tree is fixed

How can we possibly reduce search tree further?

Backtrack earlier

Backtracking occurs (only) when empty clause generated

Empty clause generated (only) by unit propagation

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

What clauses to add?

- not already in CNF
- logically entailed by CNF (or correctness lost)
- empower UP

Clause Learning

	$\Delta\|_{A}$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$
- B, C irrelevant: $\Delta|_{A} \Rightarrow \overline{X}$
- What clause would have allowed UP to derive $\overline{X}$ in level 0? $\overline{A} \vee \overline{X}$ $(A \rightarrow \overline{X})$

Clause Learning

	$\Delta\vert_A$
A, B	
B, C	B, C
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$
$\overline{A}, \overline{X}$	$\overline{X}$

$\overline{A} \vee \overline{X}$ satisfies criteria

- not in CNF
- entailed by CNF
- empowers UP

Learn clause in level 3

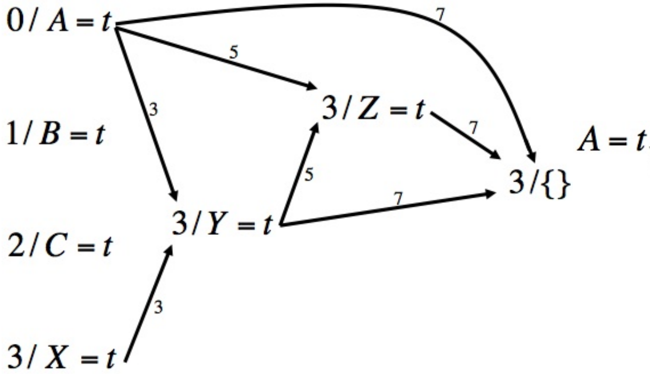
Backtrack to level 0, start over

How to learn? How far to backtrack?

Implication Graph

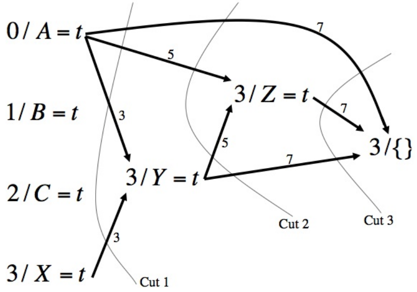
1 : A, B
2 : $\overline{B}, \overline{C}$
3 : $\overline{A}, \overline{X}, Y$
4 : $\overline{A}, X, Z$
5 : $\overline{A}, \overline{Y}, Z$
6 : $\overline{A}, X, \overline{Z}$
7 : $\overline{A}, \overline{Y}, \overline{Z}$

8 : $\overline{A}, \overline{X}$



Conflict Analysis

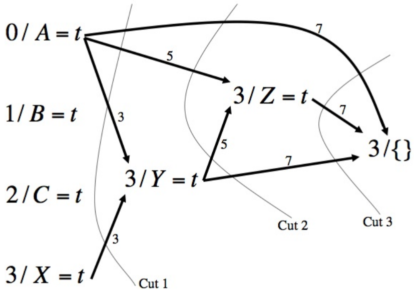
$1: A, B$
$2: \overline{B}, \overline{C}$
$3: \overline{A}, \overline{X}, Y$
$4: \overline{A}, X, Z$
$5: \overline{A}, \overline{Y}, Z$
$6: \overline{A}, X, \overline{Z}$
$7: \overline{A}, \overline{Y}, \overline{Z}$



- Cut: roots (decisions) on one side, sink (contradiction) on other
- Arrows across cut together responsible for contradiction
- Conflict set: tail points of arrows

Conflict SetConflict Clause

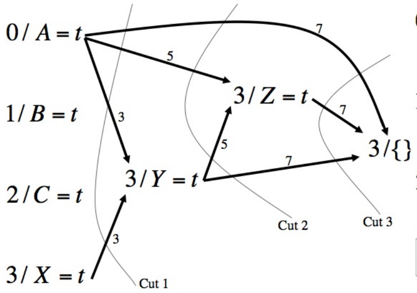
1 : A, B
2 : $\underline{B}, \underline{C}$
3 : $\overline{A}, \overline{X}, Y$
4 : $\overline{A}, X, Z$
5 : $\overline{A}, \overline{Y}, Z$
6 : $\overline{A}, X, \overline{Z}$
7 : $\overline{A}, \overline{Y}, \overline{Z}$

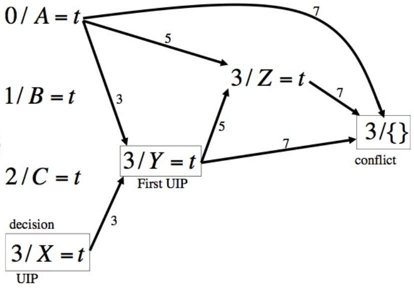


- Cut 1: $\{A, X\} \Rightarrow \overline{A} \vee \overline{X}$
- Cut 2: $\{A, Y\} \Rightarrow \overline{A} \vee \overline{Y}$
- Cut 3: $\{A, Y, Z\} \Rightarrow \overline{A} \vee \overline{Y} \vee \overline{Z}$ (existing)

  Which clause to learn?

Unique Implication Point (UIP)





Prefer shorter explanation

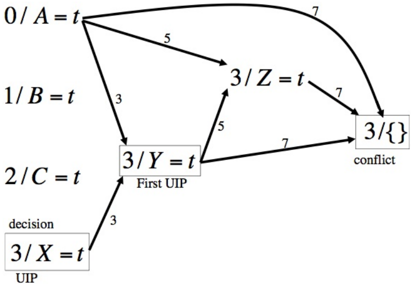
- shorter clause closer to unit, more empowering

Never need >1 node from latest level

- latest decision + history always suffices

UIP: lies on all paths from decision to contradiction

1-UIP Learning



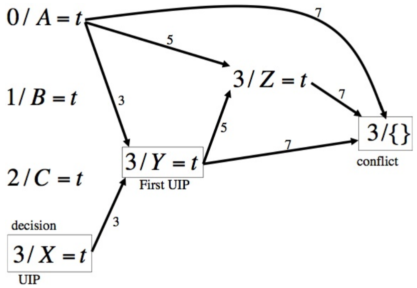
Work from sink backwards

Stop when conflict set includes a UIP, and no other nodes, of latest level: 1-UIP clause $(\overline{A} \vee \overline{Y})$

- 2-UIP, 3-UIP, . . ., All-UIP

Empirically shown effective, most common choice

Backtracking to Assertion Level



Learned clause: $\overline{A} \vee \overline{Y}$

- becomes unit ($\overline{Y}$) when erasing current level
- asserting clause: UP will assert $\overline{Y}$ (empowerment)

Backtracking to Assertion Level

Backtrack as far as possible, as long as UP remains empowered

Assertion level: 2nd highest level in learned clause, or -1 if learned clause is unit

- $A_0 \vee \overline{B}_1 \vee C_1 \vee X_4$: $aLevel = 1$
- X_4: $aLevel = -1$
- learned unit clause asserted before any decision

Empirically shown effective, most common choice

Clause Learning: Putting It Together

```
REPEAT
    IF no free variable
        RETURN SAT
    pick free variable $X$ and set either $X$ or $\overline{X}$
    IF contradiction
        IF level $< 0$
            RETURN UNSAT
        learn clause
        backtrack anywhere learned clause $\neq \emptyset$
```

- Will terminate because learned clause must be new, $|$clauses$|$ finite

Efficient Unit Propagation

Need to detect unit clauses

Naively, keep track of clause lengths: when setting X, decrement lengths of clauses that contain $\overline{X}$

- inefficient when CNF is large

Pick 2 literals to watch in each clause

- watch A, B in $A \vee B \vee \overline{C} \vee D$
- clause cannot be unit unless $\overline{A}$ or $\overline{B}$ is set
- do nothing when C or D is assigned

Scales to millions of clauses in practice

Clause Learning and Resolution

Each learning step corresponds to a sequence of resolutions

Each learned clause is the final resolvent of that sequence

Syntactic and Semantic Reasoning Combined

Searching for model and performing resolutions at the same time

Terminates as soon as either thread terminates

Solves our earlier issue with resolution (as purely syntactic method)

- Search behavior guides generation of resolvents

Addition of new clauses also influences search

- Aim for "positive feedback loop"

SAT Resources

- SAT conferences, www.satisfiability.org
- SAT competitions, www.satcompetition.org
- SAT Live, www.satlive.org
- Handbook of satisfiability

Propositional Reasoning Beyond SAT

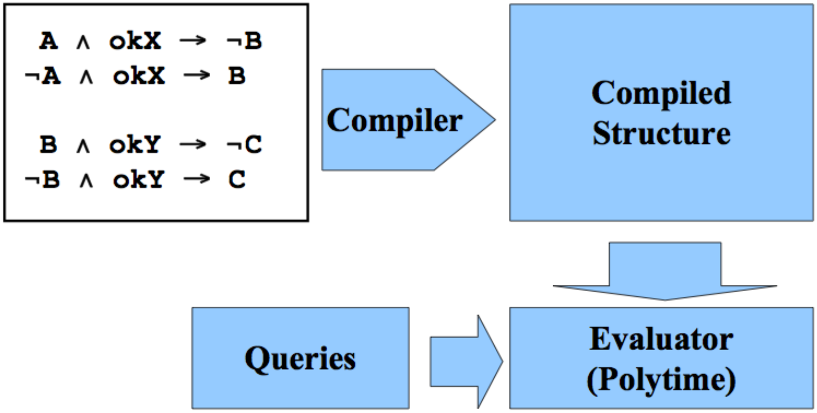
SAT answers a single query

Repeated queries on same knowledge base Δ?

- E.g., abnormality of different observations
- Each SAT query can be expensive
- Can we do some expensive work once, so that future queries can all be answered quickly?

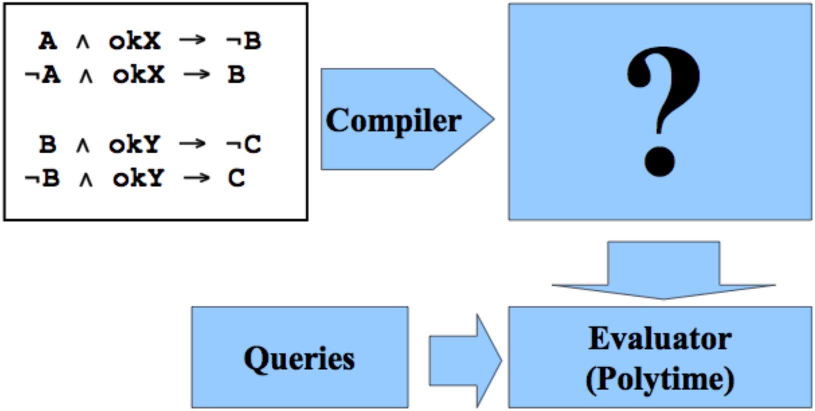
Compile Δ into *tractable* form

Multiple Queries



$A \wedge okX \rightarrow \neg B$
$\neg A \wedge okX \rightarrow B$
$B \wedge okY \rightarrow \neg C$
$\neg B \wedge okY \rightarrow C$
Compiler
Compiled Structure
Queries
Evaluator (Polytime)

What to Compile To



Target Compilation Forms

Representations of Boolean functions

Systematic space of normal forms

Satisfy different sets of properties

Support different sets of operations

Reasoning by Knowledge Compilation

Identify forms that support desired operations

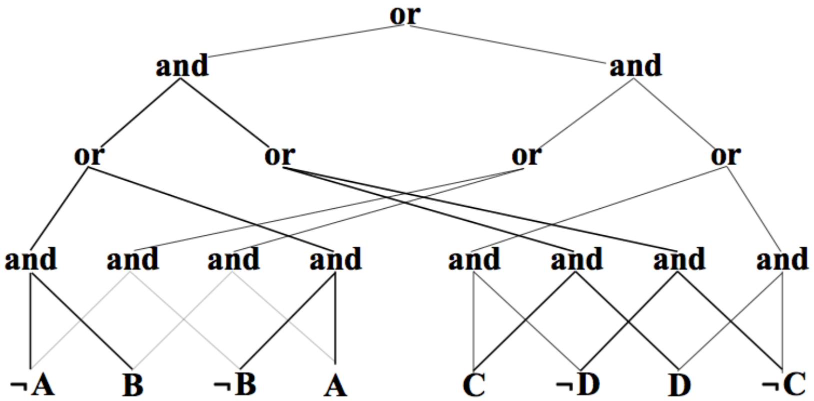
Pick most *succinct* one

Compile knowledge into target form

Perform reasoning

Negation Normal Form (NNF)

AND/OR circuit over literals



Negation Normal Form (NNF)

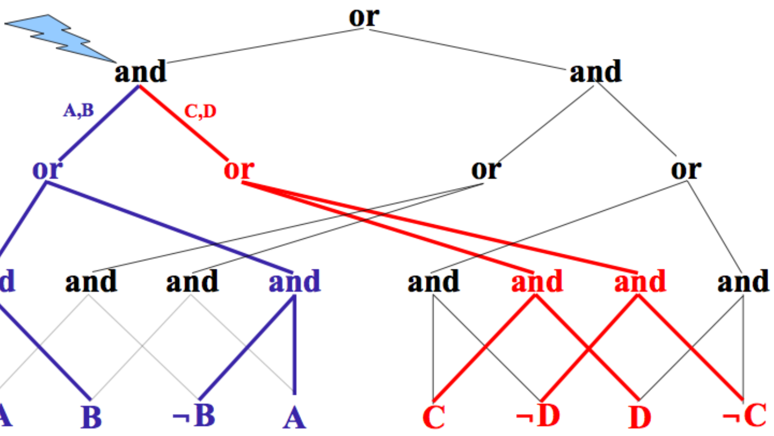
Any Boolean function can be represented in NNF (completeness)

Impose conditions/properties over NNF

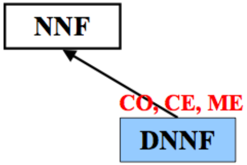
Obtain subsets/restrictions of NNF

Consider only complete subsets

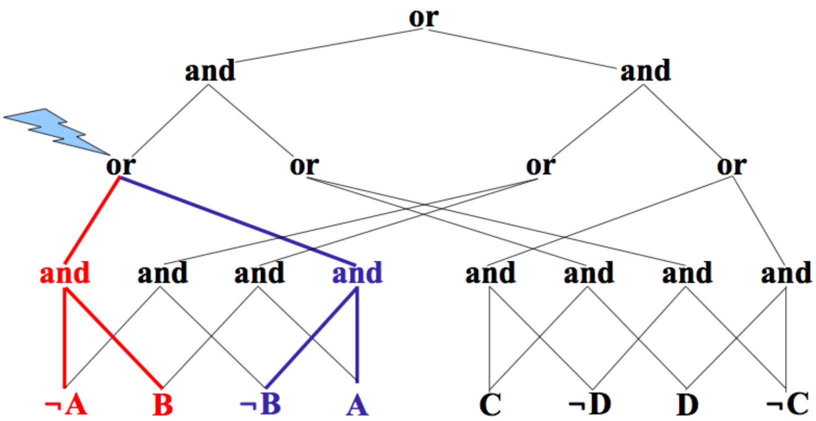
Decomposability



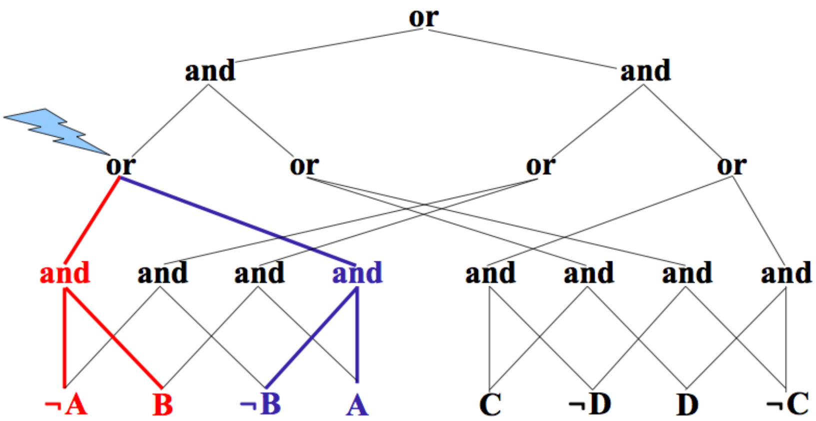
Decomposable NNF (DNNF)



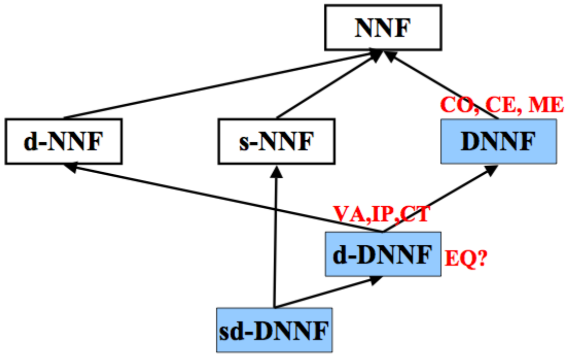
Determinism



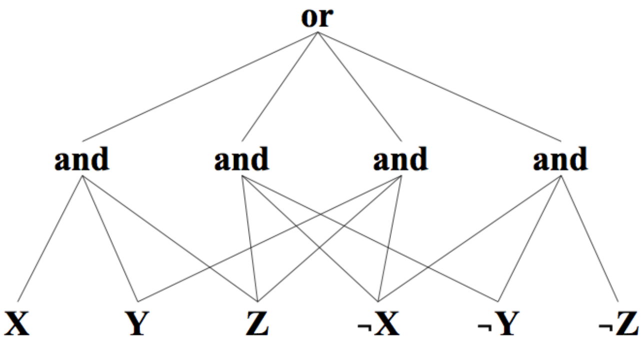
Smoothness



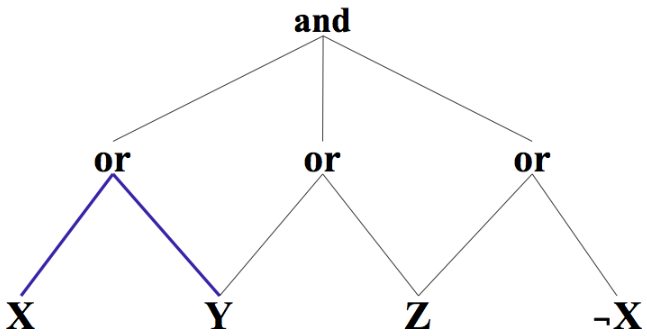
Deterministic DNNF (d-DNNF)



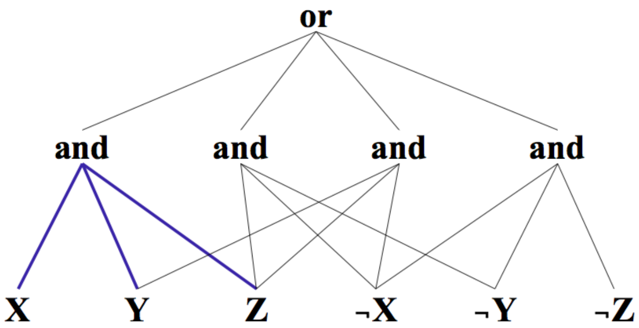
Flatness



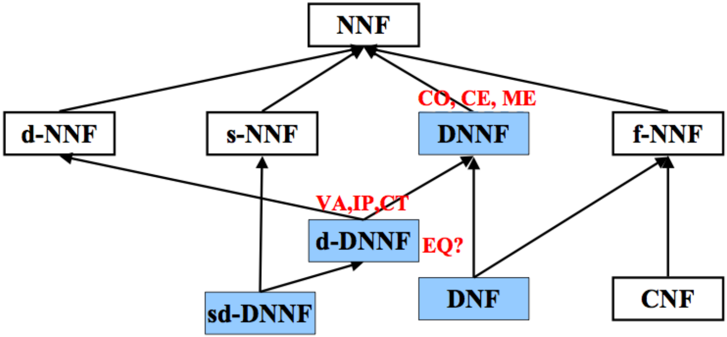
Simple Disjunction



Simple Conjunction



CNF, DNF


NNF
d-NNF
s-NNF
CO, CE, ME
DNNF
f-NNF
VA,IP,CT
d-DNNF
EQ?
sd-DNNF
DNF
CNF

Prime Implicates (PI)

Conjunction of clauses

Includes every entailed clause

No clause subsumes another

Can obtain by running resolution to saturation, removing subsumed clauses

Prime Implicants (IP)

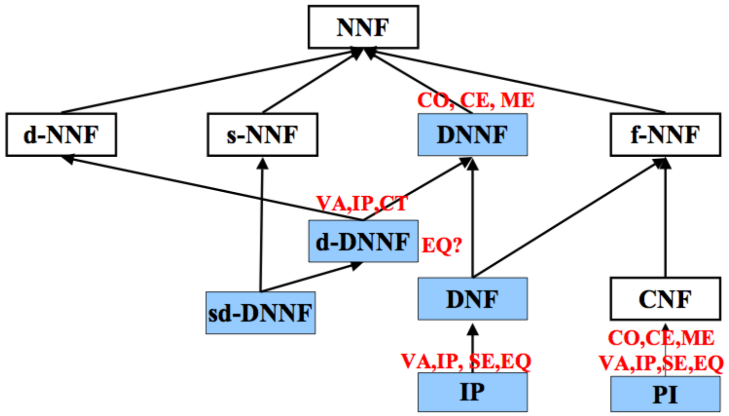
Disjunction of terms

Includes every entailing term

No term subsumes another

Can obtain by process dual to resolution

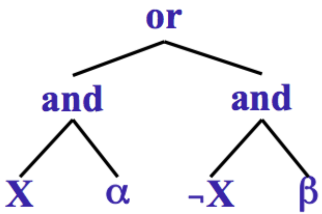
PI,IP



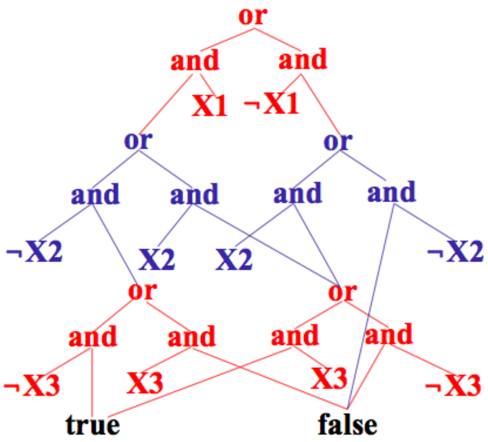
NNF
d-NNF
s-NNF
CO, CE, ME
DNNF
f-NNF
VA,IP,CT
d-DNNF
EQ?
sd-DNNF
DNF
CNF
CO,CE,ME
VA,IP, SE,EQ
VA,IP,SE,EQ
IP
PI

Decision

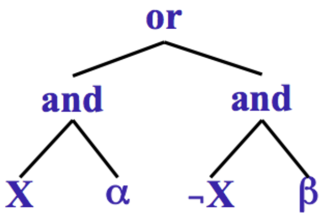
Decision node: **true**, **false**, or following fragment where α, β are decision nodes

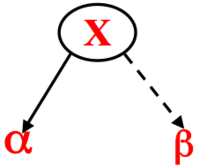


Decision

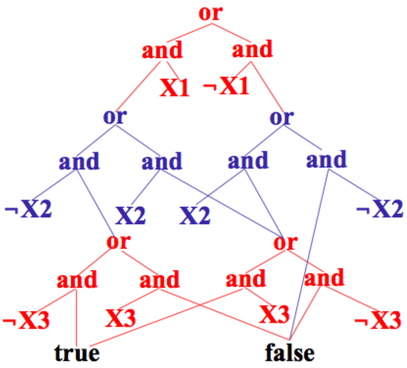

or
and
and
X1
¬X1
or
or
and
and
and
and
¬X2
X2
X2
¬X2
or
or
and
and
and
and
¬X3
X3
X3
¬X3
true
false

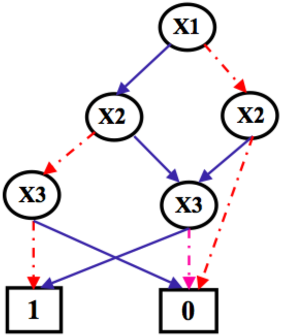
Decision Drawn Compactly



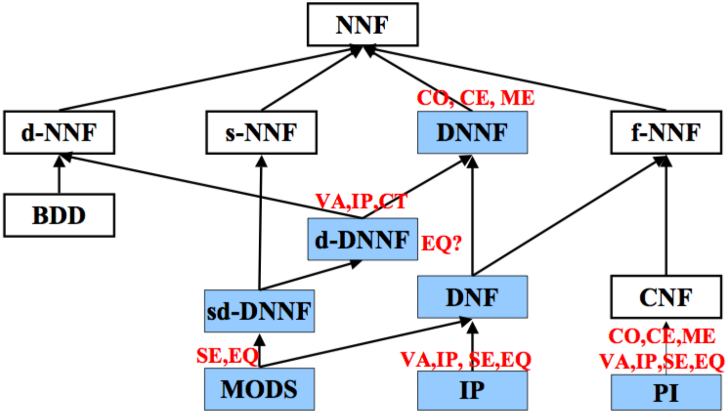


Binary Decision Diagrams (BDD)

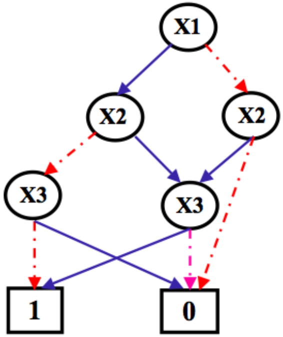




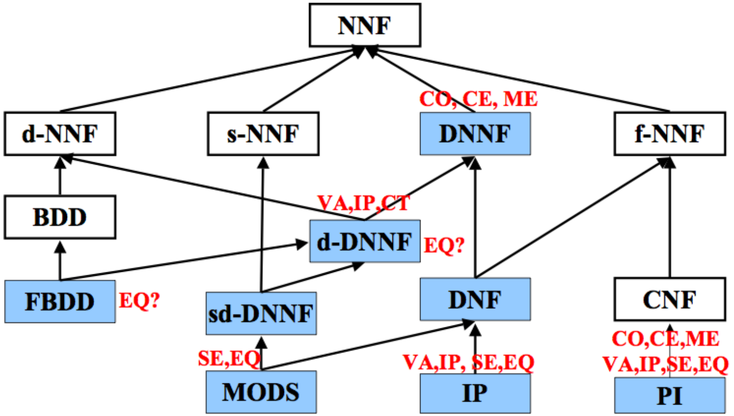
BDD


NNF
d-NNF
s-NNF
CO, CE, ME
DNNF
f-NNF
BDD
VA,IP,CT
d-DNNF
EQ?
sd-DNNF
DNF
CNF
CO,CE,ME
SE,EQ
VA,IP, SE,EQ
VA,IP,SE,EQ
MODS
IP
PI

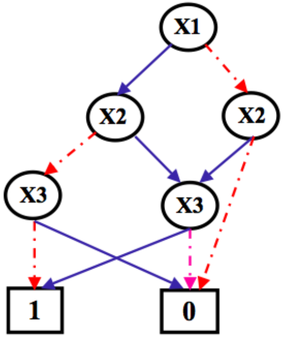
Test-once



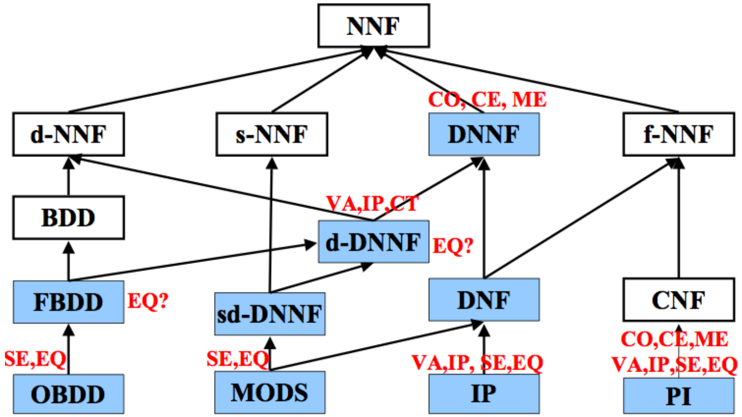
Free BDDs (FBDD)



Ordering



Ordered BDDs (OBDD)



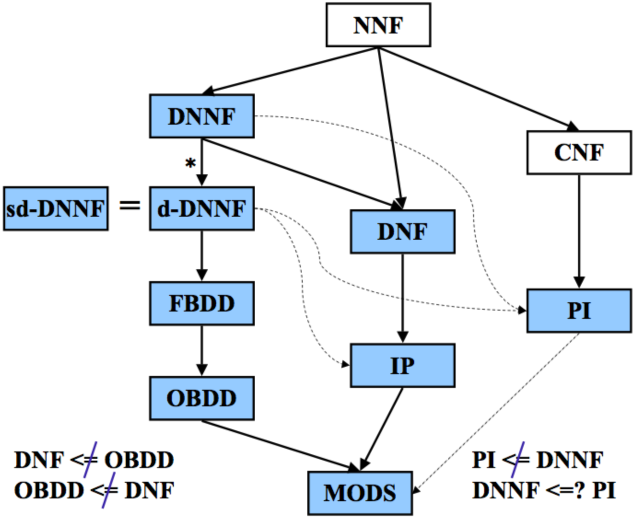
Succinctness

F *at least as succinct as* G, $F \leq G$, if

$\forall \Delta \in G$, $\exists \Delta' \in F$, $\Delta' \equiv \Delta$, $|\Delta'|$ polynomial in $|\Delta|$

F *more succinct than* G, $F < G$, if $F \leq G$, $G \not\leq F$

Succinctness



DNF $\cancel{<=}$ OBDD
OBDD $\cancel{<=}$ DNF

PI $\cancel{<=}$ DNNF
DNNF <=? PI

Further Topics

Operations supported by each form

Compiling knowledge into these forms

Reasoning with these forms

THE END